

GNU RadioとUSRPを利用した無線 通信プログラミング

グループ演習

講師： 石原 進（静岡大学創造科学技術大学院）
チュータ：坂本 浩（静岡大学大学院情報学研究科）
星川 雄大（静岡大学大学院工学研究科）

2009/10/21(水) 東北大学

グループ演習の内容

- 第1部：準備
 - GNU Radioのインストールについて
 - Python超入門
 - GNU Radioのパッケージについて
- 第2部：GNU Radio付属プログラムについて
- 第3部：GNU Radioプログラミング
 - 演習1：正弦波を送る
 - 演習2：パケットの送受信
- 付録（時間が許せば・・・）
 - モジュールプログラミングの基礎
 - 免許の取得について

第1部：準備

GNU Radioのインストール

- 現在の最新バージョンは3.2
- 最もインストールしやすいのは, Ubuntu Linux 9.04.
バイナリパッケージ有り
 - 以前のバージョンだと一手間必要
 - 他のLinuxディストリビューションだと必要パッケージをそろえるのが大変
 - Cygwinにもインストールできるらしいが...
- Ubuntu8.10でのインストール事例はWikiをご覧ください.

バイナリインストール(非常に楽)

- GNU Radio 3.2 on Ubuntu 9.04
- /etc/apt/source.list. に以下の2行を追加
 - deb http:deb http://gnuradio.org/ubuntu stable main
 - deb-src http://gnuradio.org/ubuntu stable main
- sudo apt-get update
- sudo apt-get install gnuradio gnuradio-companion
- sudo addgroup *yourid* usrp ←HWアクセスに必要

注意しましょう

- USRP2を使う場合，ギガビットイーサネットが必要.
 - 研究室に転がっている古マシンで使えないことがありますので注意.
- USRPにコマンドを送るにはroot権限が必要
 - sudo して作業する

GNU Radioに関するドキュメント

- (gnuradioモジュールの) pydocを見る
 - pydoc gnuradio.gr.クラス名
- pythonモジュールのソースを見る
 - Python 2.5:
 - /usr/lib/python2.5/site-packages/gnuradio/
 - Python 2.6:
 - /usr/lib/python2.6/dist-packages/gnuradio/
- GNU Radio 最新版のDoxgen
 - <http://gnuradio.org/doc/doxygen/hierarchy.html>

ネット上の情報源

- GNU Radio/USRP Wiki
 - <http://www.mlab.t.u-tokyo.ac.jp/~saru/usrp>
- 本家GNU Radioのページ gnuradio.org
 - チュートリアル
 - <http://gnuradio.org/trac/wiki/Tutorials/WritePythonApplications>
 - 読み物リスト
 - <http://gnuradio.org/trac/wiki/SuggestedReading>
- メーリングリストの過去ログ
 - <http://gnuradio.org/trac/wiki/MailingLists>
- Comprehensive GNU Radio Archive Network (CGRAN)
 - <http://www.cgran.org>

GNU Radioの配布パッケージ (主要なもの)

- GNU Radio 3.2 on Ubuntu 9.04の場合
 - /usr/bin/ 下記のユーティリティ群
 - gr_*.py
 - usrp_*.py usrp2_*.py
 - lsusrp
 - /usr/share/gnuradio サンプルなど
 - /usr/share/gnuradio/examples に例がある
 - /usr/share/doc/gnuradio ドキュメント
 - /usr/include/gnuradio/ ヘッダファイル群
 - /usr/lib/python2.6/dist-packages/gnuradio
 - python用のモジュール
 - /usr/lib/libgnuradio* ライブラリ本体

GNU Radioのモジュール

gr	GNU Radioのメインモジュール
usrp usrp2	USRP, USRP2のシンク, ソース, 制御モジュール
blks2	Pythonで書かれた各種モジュール: モジュレータ, デモジュレータ, フィルタなど
optfir	最適FIRフィルタ設計用のルーチン
plot_data	データプロット用の機能
wxgi	GUI用のモジュール
eng_notation	M, G等の単位を使うためのモジュール
gru	misc.

Python超入門

- GNU Radioでは, Pythonでシステムを組む
 - モジュールがあるものであれば, Pythonだけでシステムは組める
 - モジュールはC++で書かかれている. これらをPythonのインタフェースを通して利用
- Python:
 - オブジェクト指向のスクリプト言語
 - インデントが制御構造上の意味を持つ
 - 可読性が悪いプログラムを書きにくい
 - テキト一な擬似コードがそのまま動くコードになっているイメージ

Python版Hello World

やってみま
しょう

- ものすごく単純なスクリプト hello.py

```
print "Hello World" # 改行つきです
```

– 実行: python hello.py

- 単体のコマンドにするには, こう書いて...

```
#!/usr/bin/env python  
print "Hello World"
```

– mv hello.py hello

– chmod ugo+x hello

これは単に実行可能スクリプトに
拡張子をつけたくないから

- ./hello

実行

ファイルを実行可能にする

対話モードで
Python起動

数

```
prompt$ python
>>> a = 1/3  整数の割り算
>>> print a
0
>>> a = 1.0/3 実数にするなら
>>> print a  明示的に
0.333333333333333
>>>
```

複素数もサポート

```
>>> f = 1 + 2j
>>> g = f * f
>>> print g
(-3+4j)
>>> print g.real 実部
-3.0
>>> print g.imag 虚部
4.0
```

Shやperlとは違って、変数を参照するのに、
\$とか@とかという変な記号は不要.

演算子

- だいたいC言語と同じ
- ただし, C言語における && , || , !は, それぞれ, and, or, not
- ++, --は使えないが, +=, -=などはOK

文字列

```
>>> g = 'gnu radio' # シングルクォートか
>>> u = "usrp"      # ダブルクォートで囲む
>>> print g
gnu radio
>>> print print g + ' ' + u # 連結
gnu radio usrp
```

制御構造

if 条件:

```
# code code # なにもしないときは pass を書く
```

elif 条件:

```
# code code
```

else:

```
# code code
```

for x in LIST: # shのfor, perlのforeach 風

```
# code
```

```
# code
```

while 条件:

```
# code
```

```
# ループからの脱出はbreak
```

```
# ループの先頭に戻るのはcontinue
```

関数の定義

```
def fib(n): # nまでのフィボナッチ数を返す
    """Return Fibonacci series up to n."""
    result = [] 自動生成するドキュメントに反映される
    a, b = 0, 1 # リストへの代入
    while b < n:
        result.append(b)
        a, b = b, a + b
    return result  リストを戻り値として返している
```

クラス

```
# parentを親クラスとしたクラスmyclass
class myclass(parent):
    "a simple class"

# コンストラクタ（インスタンス生成時に実行）
def __init__(self):
    memval = 10.0 # メンバ変数
    # code...

# メソッドの定義
def showsomething(self):
    print "Hello again."

x = myclass() #インスタンスの生成
X.showsomething() # メソッドを呼び出し
```

例外処理

```
try:
    #例外が発生する処理
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError, (errno, strerror):
    print "I/O error:%s" % strerror
except ValueError:
    print "Could not convert data."
else:
    print i
finally:
    #例外の有無にかかわらず最後に行う処理(省略可)
```

第1部 補足

簡単な練習問題(今日はやりません)

- 1 から100までの3と7の公倍数を小さい順に印字するPythonプログラムを作れ
- クラスanimalをつくり, 何もしないクラスメソッドbarkをつくれ
 - 本当は何もしないのではなく, raise NotImplementedError するのが良い
- animalのサブクラスとしてfoxとdogをつくり, bark()を実装せよ
- リストにfox, dogのインスタンスを格納後, 順に呼び出して, bark()させよ.

文字列: 応用編

```
>>> g = 'gnu radio'
>>> print g[1] # 添え字標記
n
>>> print g[2:5] # スライス
u r
>>> print g[-2] # 後ろから2文字目
i
>>> print g[2:] # 最初の2文字以外
r radio
>>> g[2] = 'U' # これは出来ない(書き換えできない)
```

(2文字目の前から, 5文字目の前)

g	n	u		r	a	d	i	o
0	1	2	...	5				

リスト

リストには、何でも格納できる。リストの要素の異なってもよい。

```
>>> myList = ['warp', 'sora', 100, 0]
>>> myList[0]
'warp'
>>> myList[1, 3] # スライス
'sora', 100
>>> myList[1, 3] = 'usrp', 6 # スライスを書き換え
>>> myList
['warp', 'usrp', 6, 0]
>> myList.append(25) # リストに25を追加
>>> myList
['warp', 'usrp', 6, 0, 25]
# このほか、len(), sort()など様々なリスト操作メソッドがある
```

関数range()

range(): 任意の要素数の等差数列の入ったリストを生成する

```
>>> range(5)
```

```
[0, 1, 2, 3, 4]
```

```
>>> for i in range(2, 20, 3): # 範囲と増分を指定
```

```
...     print i
```

```
...
```

```
2
```

```
5
```

```
8
```

```
11
```

```
14
```

```
17
```

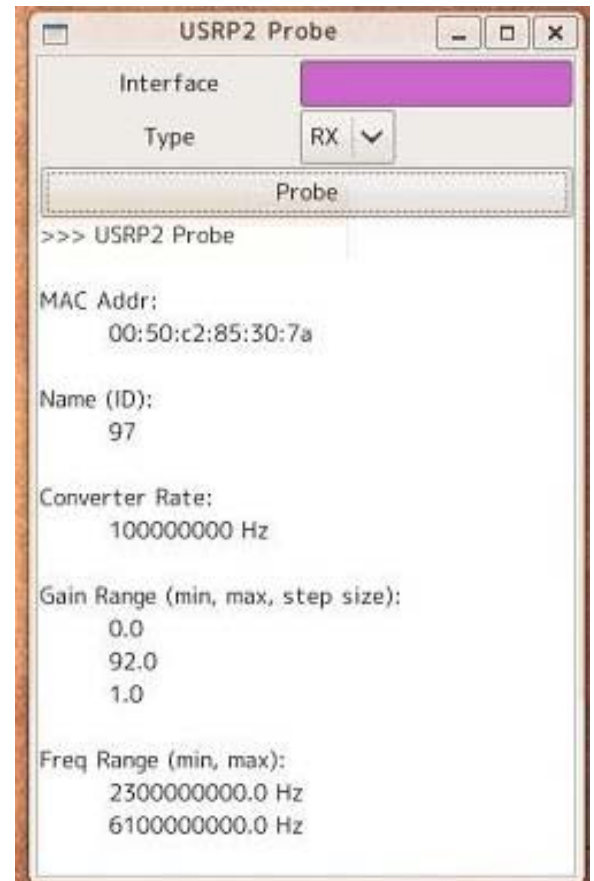
```
>>>
```

第2部 : GNU Radio付属プログラムについて (/usr/bin 以下)

- usrp2_probe
 - DaughterBoardの接続チェック
- usrp2_fft.py
 - スペクトラムアナライザー

usrp2_probe

- 起動
 - `sudo usrp2_probe`
- 使い方
 1. Interfaceに"eth0"を入力
 2. TxかRxを選択
 3. 「Probe」ボタンを押す
- 結果
 - USRP2とDaughterBoardの情報を表示

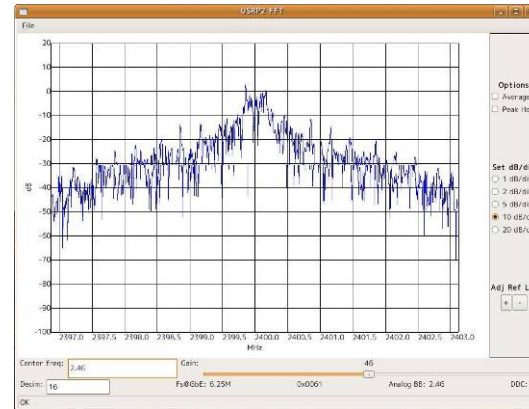
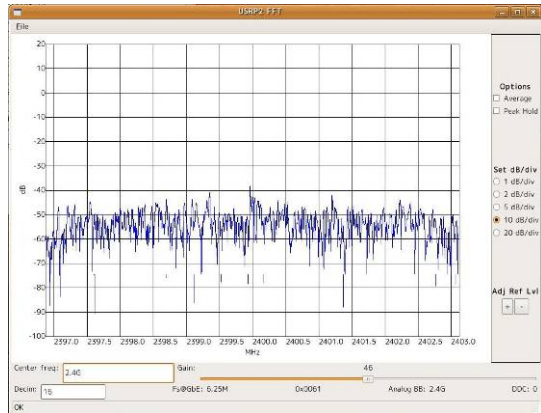


usrp2_fft.py

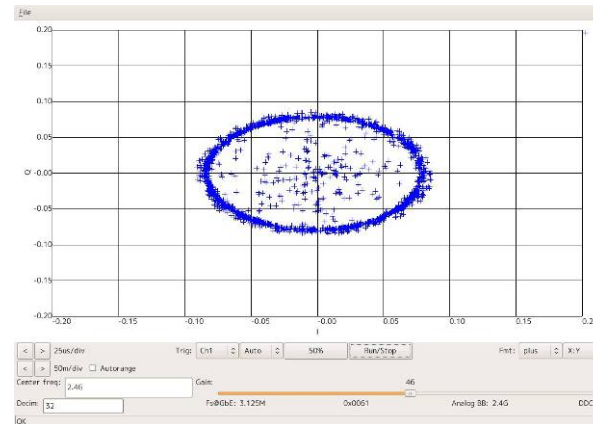
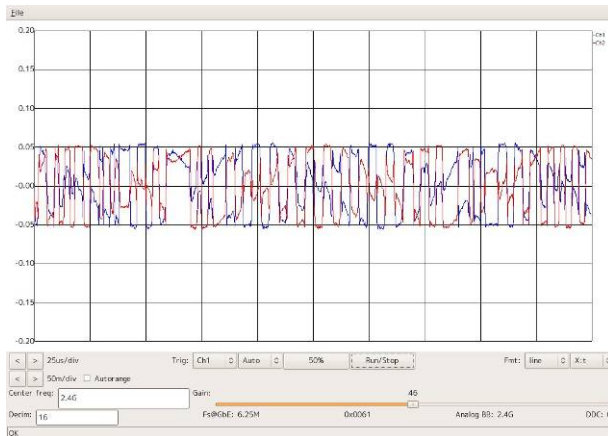
- スペクトラムアナライザー
- 使い方
 - `sudo usrp2_fft.py [options]`
 - option例
 - オシロスコープ表示 `-S`
 - ウォーターフォール表示 `-W`
 - 周波数 ⇒ `-f Freq ex) -f 2.4G`
 - ゲイン ⇒ `-g Gain ex) -g 17`
 - decimation ⇒ `-d Decim ex) -d 16`

usrp2_fft.py動作例

`usrp2_fft.py -f 2.4G`



`usrp2_fft.py -f 2.4G -S`



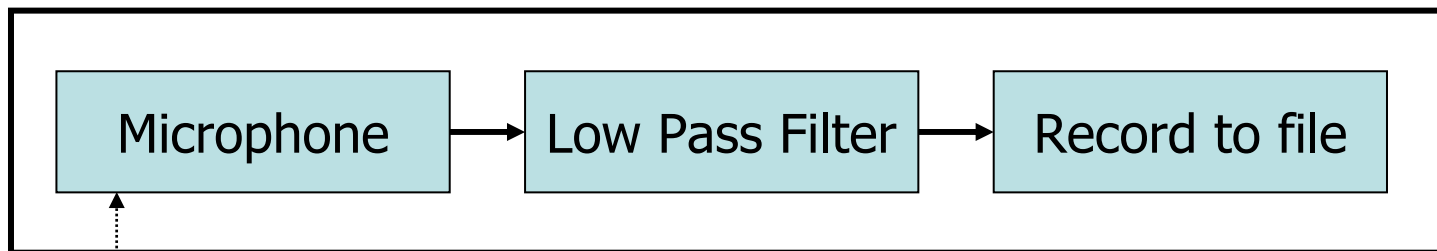
オシロスコープ表示

第3部 : GNU Radio プログラミング

GNU Radioでのプログラミング

- GNU RadioでのPythonプログラミング
= フローグラフを作ること
 - C++で記述されたブロックを, Pythonの記述によって, 接続していく

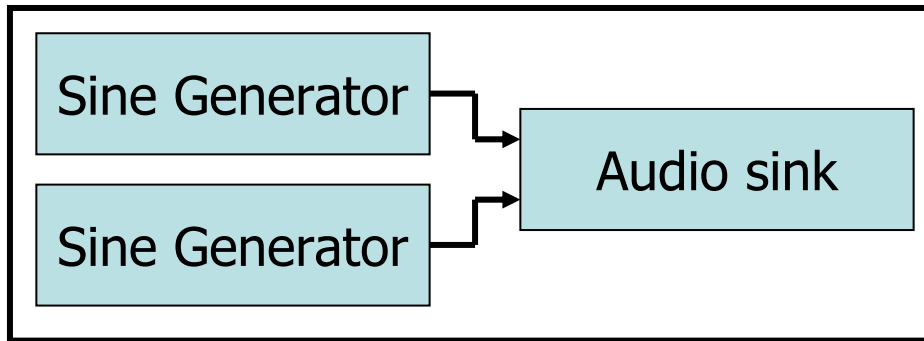
Low-pass filtered audio recorder



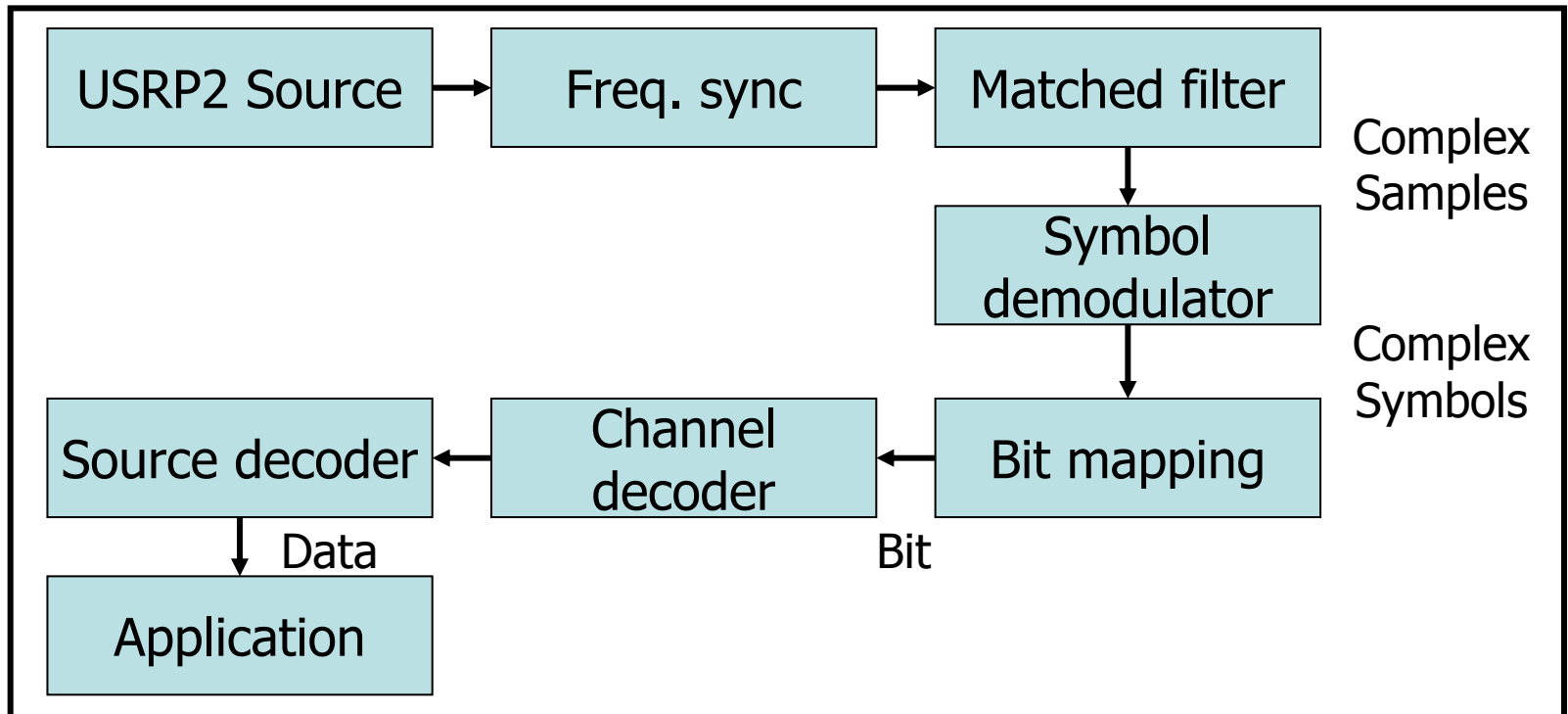
C++で書かれている

フローグラフの例

Dial Tone Generator

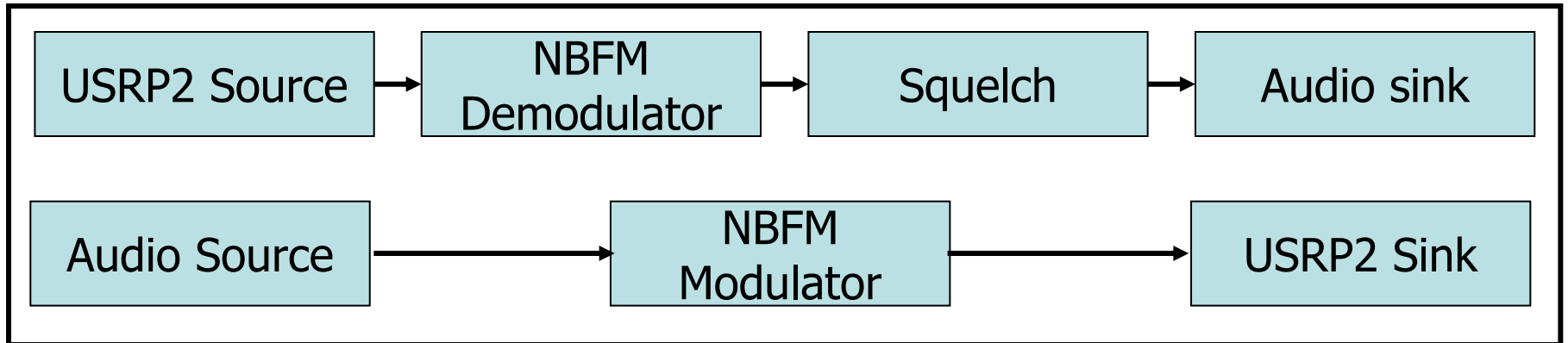


QPSK Demodulator



独立した複数のフローがあるグラフ

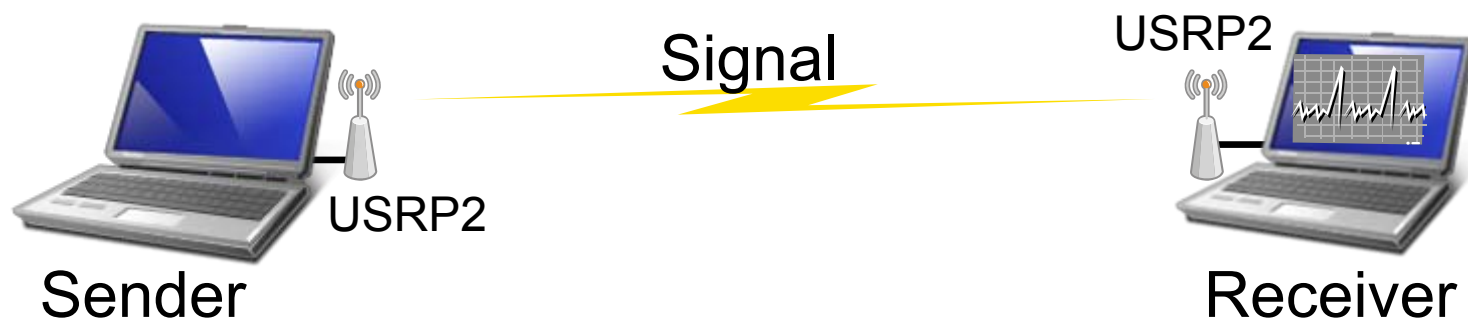
Walkie Talkie (いわゆるトランシーバ)



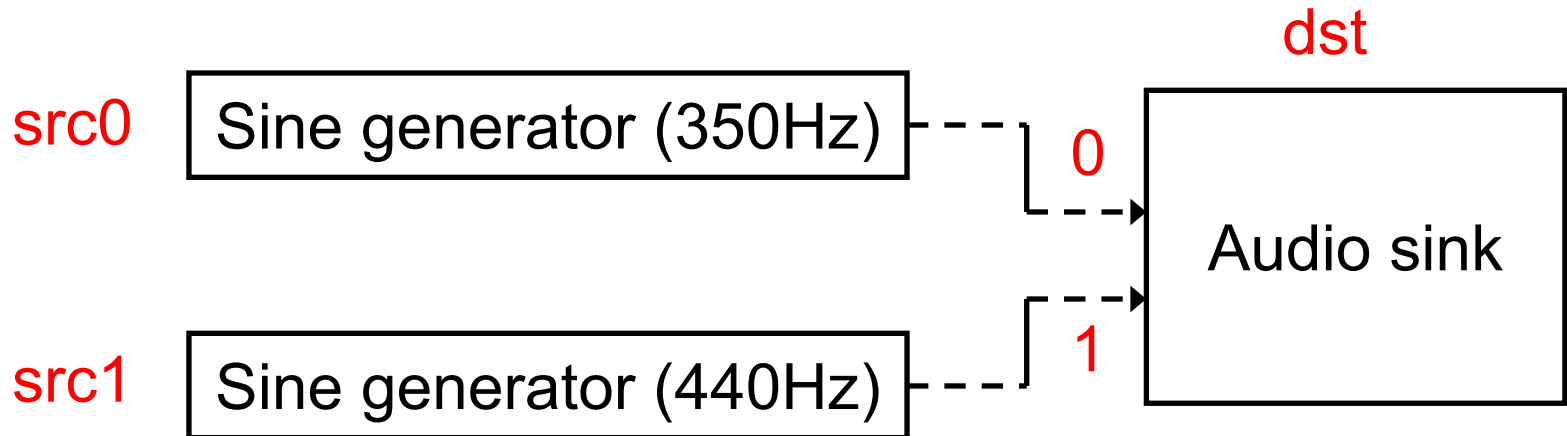
フローグラフ以外にこれらそれぞれの動作の停止・再開を制御するコードが必要

演習1: 正弦波を送る

- USRP2を使って実際に信号を送信し、信号が送信されているかを確認
 - 正弦波をUSRP2で送信
 - 受信側でスペクトラムアナライザを用いて信号が送信されているかを確認



プログラム例: Dial tone Generator



- 信号を発生させるモジュール
 - `gr.sig_source_f (sampling_rate, 波形, 周波数, ampl)`
- オーディオの受け口
 - `audio.sink (sampling_rate, "")`

前ページのグラフに相当するPythonコード

```
#!/usr/bin/env python
from gnuradio import gr # importはC/C++で言う#includeと同じ
from gnuradio import audio

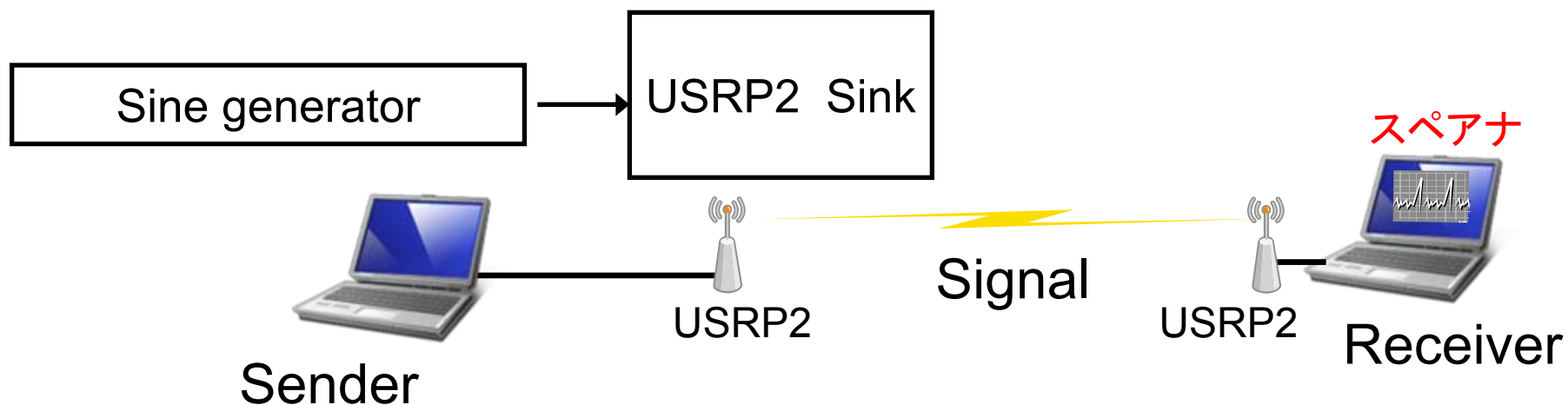
class my_top_block(gr.top_block): # gr.top_blockを継承
    def __init__(self): # コンストラクタの定義
        gr.top_block.__init__(self) # 親クラスのコンストラクタを呼び出す
        sampling_rate = 32000
        ampl = 0.1
        # build graph
        src0 = gr.sig_source_f(sampling_rate, gr.GR_SIN_WAVE, 350, ¥
                               ampl)
        src1 = gr.sig_source_f(sampling_rate, gr.GR_SIN_WAVE, 440, ¥
                               ampl)
        dst = audio.sink (sampling_rate, "")
        self.connect(src0, (dst, 0)) # ブロック間を接続
        self.connect(src1, (dst, 1))

if __name__ == '__main__':
    try:
        my_top_block().run()
    except KeyboardInterrupt:
        pass
```

run(): flow graphを動かす最も単純な方法
start()を呼び出し、その後wait() でflow
graphを実行し続ける

USRP2でSIN波を送る:手順

1. 信号を生成
2. USRP2の受け口を用意
3. 信号をUSRP2の受け口に渡す
4. 受信側で信号を確認
 - スペクトラムアナライザプログラムを使用



Step 1: 信号を生成

- 以下の関数を用いて, 信号生成ブロックを製作
 - `gr.sig_source_c` (sampling_rate, 波形, 周波数, ampl)
- 波形は`gr_sig_source_waveform.h`で定義されている
 - SIN波 ⇒ `gr.GR_SIN_WAVE`
 - COS波 ⇒ `gr.GR_COS_WAVE`
 - 矩形波 ⇒ `gr.GR_SQR_WAVE`
 - 三角波 ⇒ `gr.GR_TRI_WAVE`
 - のこぎり波 ⇒ `gr.GR_SAW_WAVE`

Step 2: USRP2の受け口を用意

- USRP2用のモジュールをインポート
 - `from gnuradio import usrp2`
- USRP2送信機のブロックを作成
 - `usrp2.sink_32fc()`メソッドでインスタンスを生成
 - 第1引数: USRP2を接続しているインタフェースを表す文字列 ex) "eth0"
 - 第2引数: USRP2のMACアドレスを表す文字列 ex) "00:19:B9:25:CE:D7"

Step 3: 信号をUSRP2に渡す

- `gr.top_block.connect()`を用いて, 信号生成器とUSRP2の受け口を接続

```
src = gr.sig_source_c(sampling_rate, WAVE_form, ¥  
                      frequency, ampl)  
self.u = usrp2.sink_32fc(interface, mac_addr)  
# ex) interface="eth0" mac_addr = "00:AB:CD:00:FC:8W"  
# ※引数を指定しなければ初期設定が使われる 通常はなにもしなくてもOK  
  
self.connect(src, self.u)
```

Sampling_rate: 6250000

(=6.25MHz: USRP2のDAC最大能力=100Mシンボル/秒 これを16個ごとに間引き)

WAVE_form: `gr.GR_SIN_WAVE` など

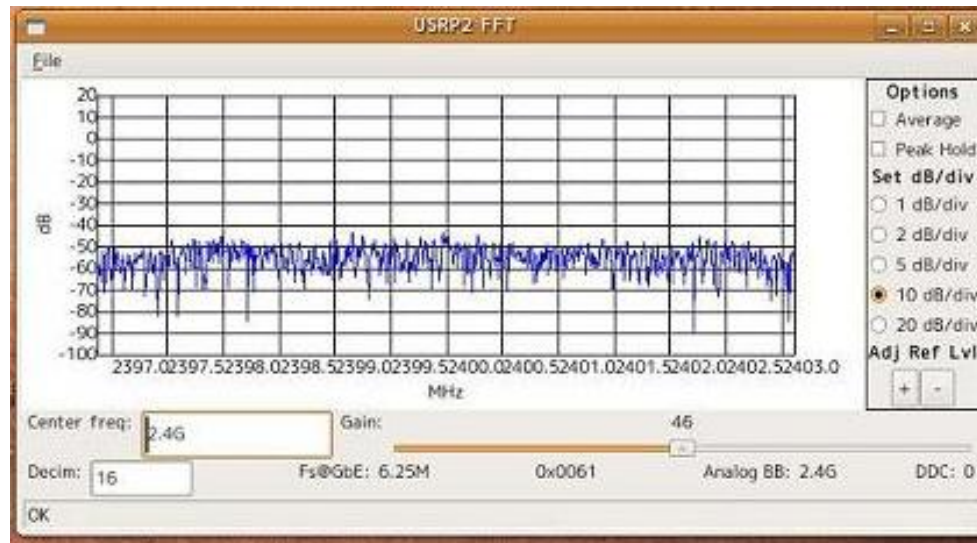
frequency: Hzで指定(例 5110000000 (=5.110GHz))

補足: USRP2-PC間のデータ送信について

- USRP2のDAC, ADCの処理能力:
それぞれ100Mサンプル/秒
- USRP2-PCでのやりとりでは, 1サンプル=32bits
(IQ 各16bits)
- ただし32bit/サンプル * 100Mサンプル/秒 =
32Gbit/s > 1Gbit/s
→Gigabit Ethernetで送信できない
- そこで間引く→decimation (interpolation) =4~512
とする
 - decimationは4の倍数とする
 - 4にするとPCの負荷がかなり厳しいが...
 - 帯域25MHz確保→802.11b,g OK
 - Ethernet上のデータ送信量 800Mbit/s以上

Step 4: 受信側で確認

- fft_sink.py(スペクトラムアナライザ)を用いて送信側が発信する信号の周波数帯を監視
 - fft_sink.pyはGNU Radioに標準で入っている
- 送信した周波数でスペクトルが確認できたらOK



雛形プログラム

```
#!/usr/bin/env python
from gnuradio import gr
from gnuradio import usrp2

class my_top_block(gr.top_block):
    def __init__(self):
        gr.top_block.__init__(self)
```

ここに自分のコードを書く

```
if __name__ == '__main__':
    try:
        my_top_block().run()
    except KeyboardInterrupt:
        pass
```

拡張: コマンドラインのオプション

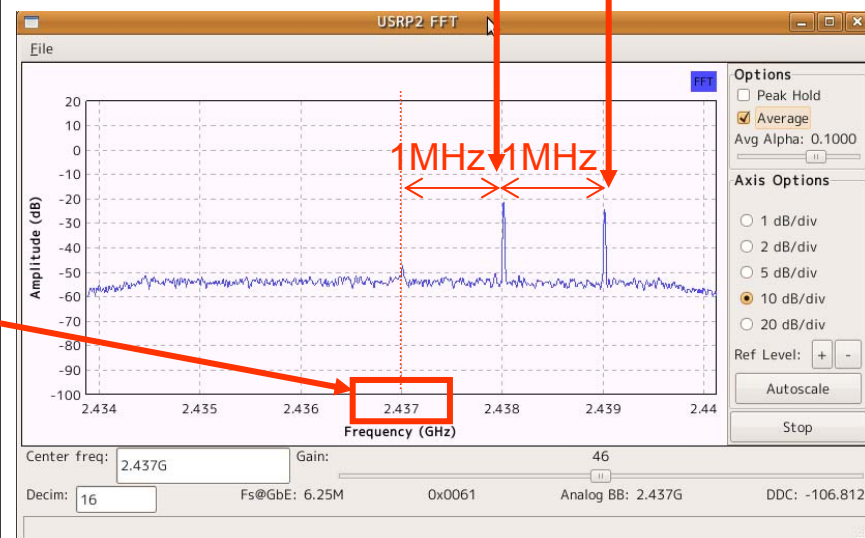
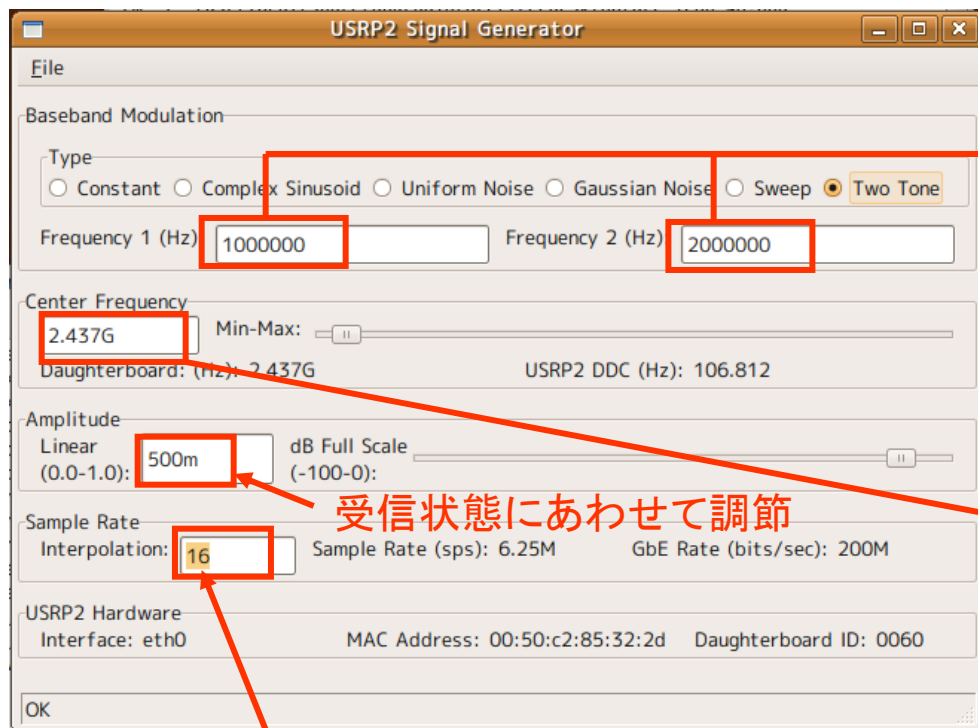
- コマンドラインのオプションで, 周波数, サンプリングレートなどを指定可能にする

ex) `python yourmodule.py -f 5.110G`

```
#!/usr/bin/env python
# "6M" "2.4GHz"などの表記を使用可能にする
from gnuradio.eng_option import eng_option
# コマンドラインオプション使用のためのモジュールをインポート
from optparse import OptionParser

class yourmodule(gr.top_block):
    # Parse options
    parser = OptionParser(option_class=eng_option)
    parser.add_option("-f", "--freq", type="eng_float",
                    default=5110*1e6, help="set freq. to FREQ")
    (options, args) = parser.parse_args()
    src = gr.sig_source_f (sampling_rate, wave_form,
                          options.freq, ampl)
```

より賢い信号送信テストプログラム /usr/bin/usrp2_siggen_gui.py



16→サンプリングレート6.25Msps(=100Msps/16)
4がGbEの性能からして限界. CPU消費も大きい

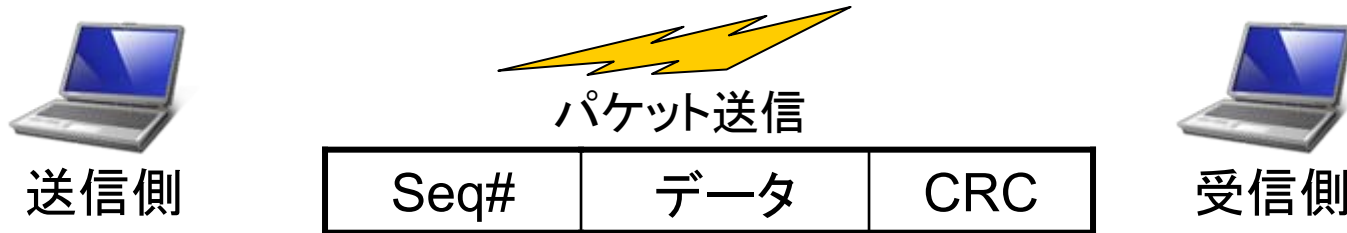
usrp2_siggen_gui.pyの画面

usrp2_fft.pyの画面
(-Wオプションを使っても面白いです)

演習2: パケットの送信

- 送信・受信を試みよう
 - グループ間で衝突しない中心周波数を選んだり、あえて衝突させるなどしてみましょう。
- 実動コードで確認しよう
 - パケットにデータをまとめるには？
 - 誤り検出符号を加えるには？
 - 変調方式を選ぶには？

パケット送受信サンプルプログラム (/usr/share/gnuradio/examples/digital/*)



- プログラム

- 送信側: `benchmark_tx.py`
- 受信側: `benchmark_rx.py`

デフォルトでは、パケット番号について、パケット番号の下位1バイトが並んだパケットが送られる

- 実行方法

- `cd /usr/share/gnuradio/examples/digital`
- 送信: `./benchmark_tx.py [options]`
- 受信: `./benchmark_rx.py [options]`
- [options]: 周波数、変調方式などを指定
 - 例) `benchmark_tx.py -f 5.110G -m dbpsk`

benchmark_tx.py の主なオプション

- **-m** 変調方式 (例 **-m dpsk**)
cpm, d8psk, qam8, dbpsk, dqpsk, gmsk [default = gmsk]
- **-s** パケットサイズ[バイト] (例 **-s 512**)
[default = 1500]
- **-f** 搬送波周波数[Hz] (例 **-f 5.110G**)
- **-r** ビットレート[bps] (例 **-r 64000**)
[default = 100000.0]
- **--tx-amplitude**=増幅率
(例 **-tx-amplitude 0.5**)
[default = 0.25] 0以上1未満
- **-S** シンボルあたりのサンプル数 (例 **-S 4**)
[default = 2]
- **-h** ヘルプ (もっといろいろなオプションがあります)

benchmark_rx.py の主なオプション

- **-m** 変調方式 (例 **-m dpsk**)
dbpsk, d8psk, dqpsk, gmsk [default = gmsk]
- **-f** 搬送波周波数[Hz] (例 **-f 5.110G**)
- **-r** ビットレート[bps] (例 **-r 64000**)
[default = 100000.0]
- **-h** ヘルプ (もっといろいろなオプションがあります)

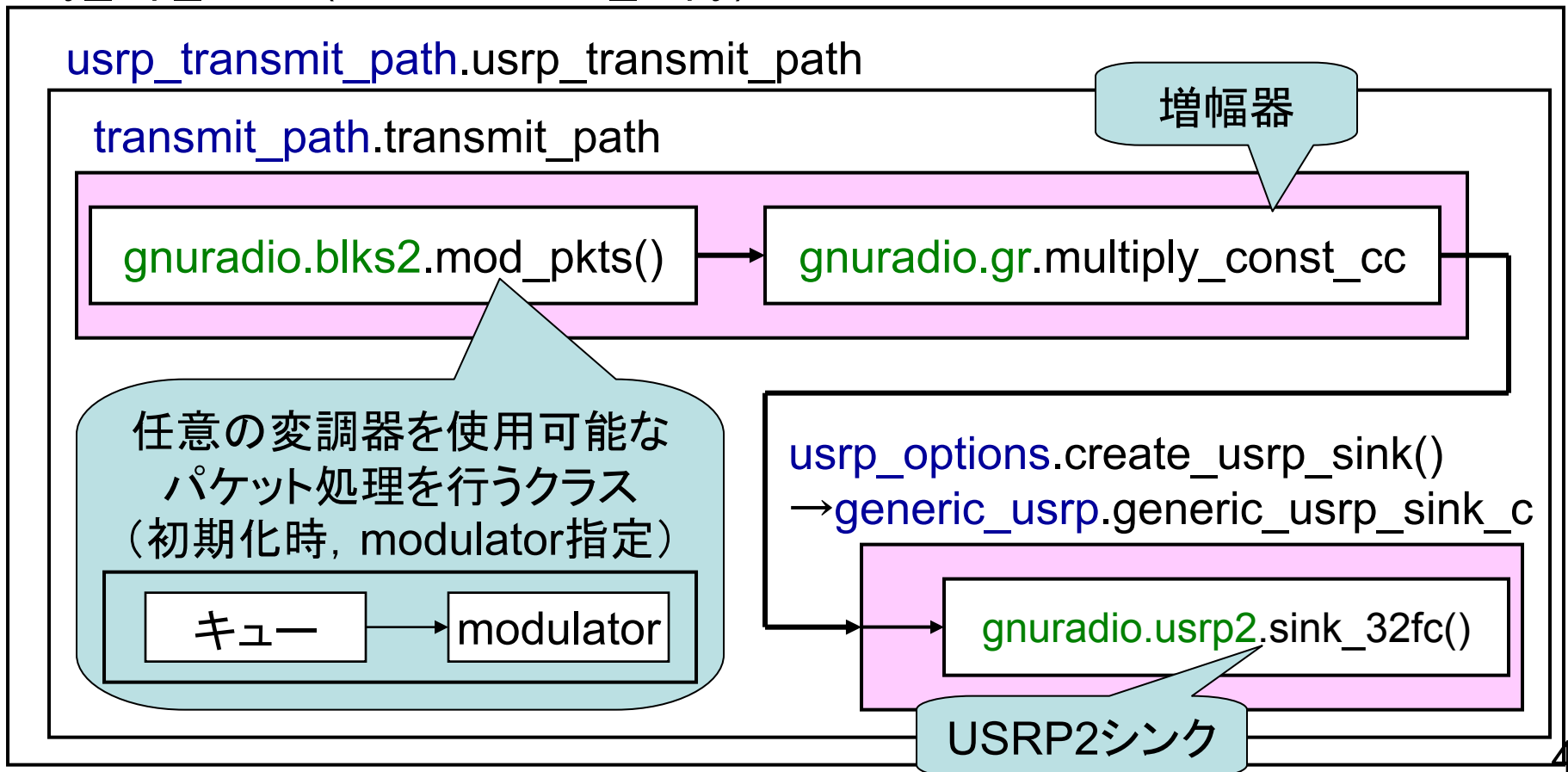
送信側プログラムの構造

benchmark_tx.pyで使うフローグラフを作るために、多くのモジュールを利用

青字: benchmark_tx.pyと同じディレクトリにある他のモジュール

緑字: GNU Radio標準のモジュール

my_top_block (in benchmark_tx.py)



パケットデータの生成過程(1/2)

1. バイト列 (Pythonでは文字列として扱う) として, ペイロードデータを作る.

benchmark_tx.pyより抜粋. コメント追加

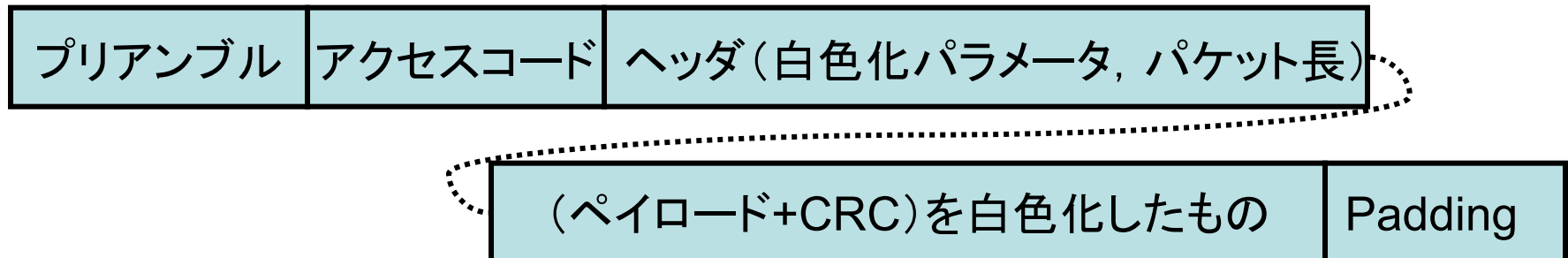
```
# パケット番号 (pktno) の下位8ビットを (ペイロード長-2) 個つなげる
data = (pkt_size - 2) * chr(pktno & 0xff)

# (略)

# 先頭2バイトにパケット番号(下位16ビット)を入れる
# struct.pack() の第1引数の意味
#   !は, ネットワークバイトオーダー(ビッグエンディアン)
#   Hはunsigned short
payload = struct.pack('!H', pktno & 0xffff) + data
```

パケットデータの生成過程(2/2)

2. ペイロードデータを(何ステップか踏むが)最終的に `gnuradio.blks2.mod_pkts.send_pkt()` に渡す
3. `gnuradio.blks2.mod_pkts.send_pkt()` では, `gnuradio.packet_utils.make_packet()` にペイロードデータを, 変調方式のパラメータと共に渡す
Pythonソース:
`/usr/lib/python2.6/dist-packages/gnuradio/blks2impl/pkt.py`
4. `gnuradio.packet_utils.make_packet()` で, 以下の構成のパケットを作る
Pythonソース:
`/usr/lib/python2.6/dist-packages/gnuradio/packet_utils.py`



コードリーディングのヒント

- `benchmark_tx.py`, `benchmark_rx.py`に関連するコードを読むことで、多くの技法を学ぶことができます。
- CRCの計算: `gnuradio.gr.crc32()`
- 変調器・復調器のコード
 - `/usr/lib/python2.6/dist-packages/gnuradio/blks2impl/d8psk.py`, `dqpsk.py`, `ofdm.py`などがあります。
- 2バイト以上の数値データを扱う場合、バイトオーダーの処理に注意。ネットワークバイトオーダーにそろえる。

謝辞

- チュートリアル実施にあたり, 下記の皆様の
他多くの皆様にご協力をいただきました。
ありがとうございました。
 - 藤井威生 先生(電気通信大学)
 - Minseok KIM 先生(東京工業大学)
 - 石橋 功至先生(静岡大学)
 - 電子情報通信学会アドホックネットワーク研究会
幹事の皆様

<付録>
GNU Radio プログラミング
～モジュール偏～

<参考>

<http://www.swig.org/>

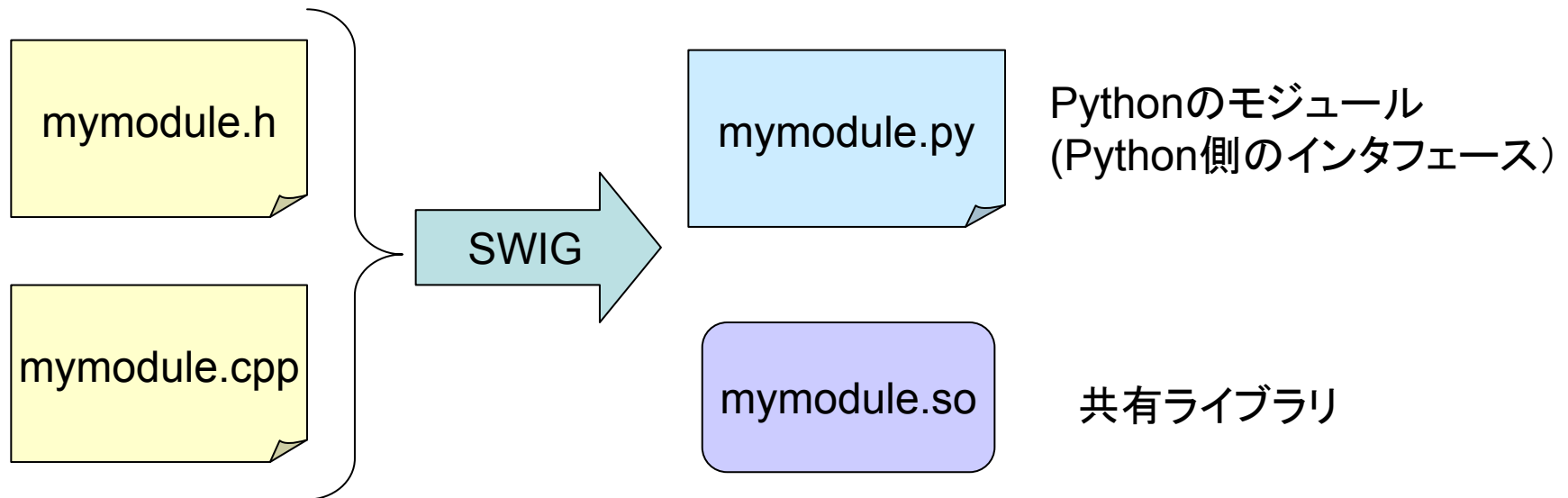
Gnu Radio/USRP Wiki コンテンツ(ブロックの作り方)

はじめに

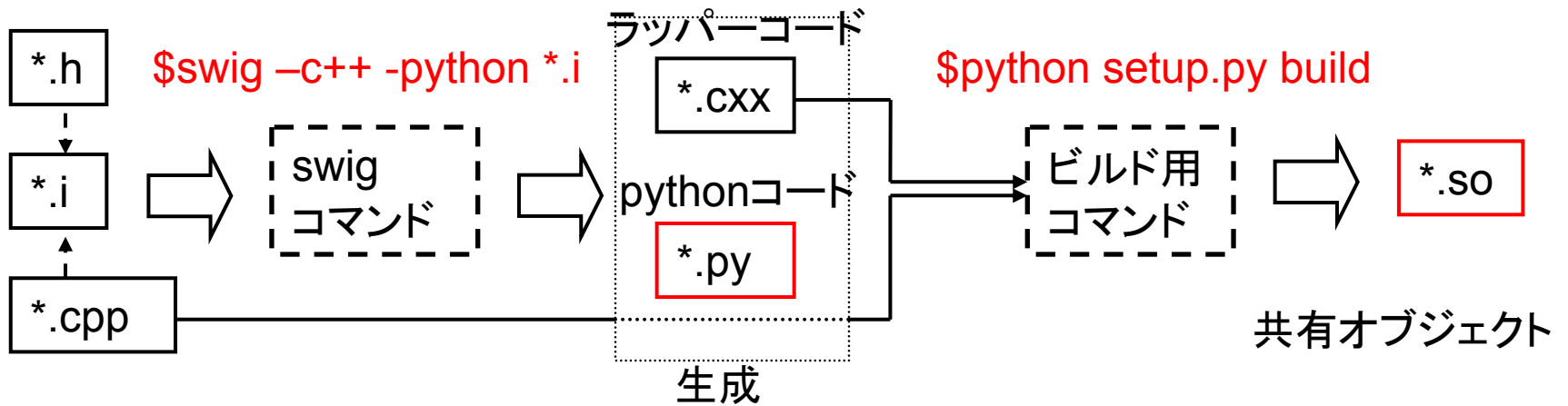
- GnuRadioのモジュールは全てC++で記述
 - モジュール自体もプログラミング可能
- C++のライブラリをPythonなどのスクリプト言語で使用するためにはSWIGを使用
 - **S**implified **W**rapper and **I**nterface **G**enerator
- この演習の目的
 - SWIGの動作を理解する
 - gnuradioに自分のモジュールを追加

SWIGってなに？

- Simplified **W**rapper and **I**nterface **G**enerator (**SWIG**)
- C/C++で書かれたライブラリをPythonなどのスクリプト言語や, Javaなどから利用するためのグルーコードを生成



C++コードからPythonモジュールの生成まで



Step1/4: SWIGファイル, ヘッダファイルを用意

booster.i (SWIGファイル)

```
%module booster

%{
#include "booster.h" インクルードするヘッダファイルを指定
%}

%include "booster.h" インクルードする関数, 変数を指定
(ヘッダファイル内の関数, 変数全て)
```

booster.h

```
class booster {
public:
    booster();
    float boost();
};
```

Step 2/5 C++ソースコードの用意

booster.cpp

```
#include <iostream>
#include "booster.h"

// コンストラクタ
booster::booster()
{
    // なにもしないことにする
}

// 入力を2倍にして出力するメソッド
float booster::boost(float input)
{
    float output;           // 戻り値の格納用
    output = input * 2;     // 引数で与えられた値を2倍に
    return (output);
}
```

Step 3/5 Pythonの拡張モジュールの ビルド用ファイル(setup.py)を書く

```
# setup.py
from distutils.core import setup, Extension

module1 = Extension('_booster', ¥
                    sources = ['booster.cpp', booster_wrap.cxx'], ¥
                    libraries = ['stdc++'])
# Extension() 拡張モジュールオブジェクトをつくるメソッド
# 第1引数 モジュール名
# 第2引数 ソースファイル
#          (swigで作ったラッパーファイルと元のC++ソース)
# 第3引数 標準C++のランタイムライブラリを使うことを明示

# 追加するパッケージのセットアップ (モジュール名の前に '_'をつける)
setup (name = '_booster', ext_modules = [module1])
```

Step 4/5 ビルド

- swigコマンドを用いてラッパー用コードを作り, setup.pyでモジュールとしてビルド・インストール

```
$ swig -c++ -python booster.i ← _wrap.cxx, .pyファイル作成  
$ python setup.py build ← buildディレクトリ以下にファイル作成  
$ python setup.py install ← build/lib以下を  
                          インストールディレクトリにコピー
```

- 以上で*.pyファイルと*.soファイルが作成できインストールが完了する

Step 5/5 使ってみる

```
# import our "booster" module
import booster

# make an instance
myBooster = booster.booster()

a = 10
# use "boost()" method
b = myBooster.boost(10)

print a, b
```

GNU Radio環境に 自分のモジュールを組み込むには

- swigコマンドで作成したbooster.pyファイルを適切な場所にコピーする
 - /usr/lib/python2.6/dist-packages/gnuradio/
- 呼び出し方法
 - `from gnuradio import "module_name"`
- 共有ライブラリ `_booster.so` はビルドのインストールの際に適切な位置に自動でコピーされる

電波を送信する

静岡大学

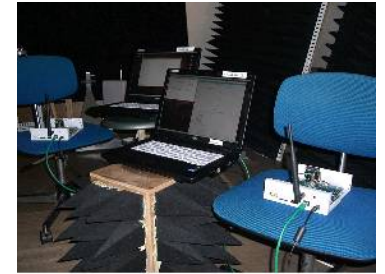
渡辺 尚

2009/10/21

無線信号を送信する方法

- 電波暗室の利用

- 利点: 比較的楽に実験が可能, 理想状態の実験可
- 欠点: 設備の借用, 物理サイズの制限を受ける



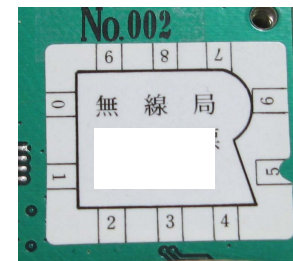
電波暗室

- 技術基準適合証明の取得

- 利点: 免許がなくても実験が可能 (ISMバンド)
- 欠点: 他局からの干渉あり, 技適検査, 費用
- 参考: 木崎一廣: "技術基準適合証明の取得体験記", RFワールド, No.6, CQ出版, 2009.6 (トラ技増刊)

- 特定実験試験局免許の取得

- 利点: 他局からの干渉なし
- 欠点: 免許, 無線設備点検, 費用, 時間
- 次ページ以降で紹介



- cf. アンテナ端子をケーブルで直結

- 利点: 確実に通信可能 テストに適する
- 欠点: キャリアセンスは不可



特定実験試験局

- 特定実験試験局とは
 - 総務省 電波利用に関する制度
 - <http://www.tele.soumu.go.jp/j/sys/spexp/index.htm>
 - 周波数等をあらかじめ公示することにより短期で免許処理が可能となる実験局の制度
 - 予め決められている周波数の範囲から特定の周波数の使用を許可
 - 利点:他の無線局から, および他の無線局への干渉を防ぐことができる. MACなどISMバンドでは難しいものを実験できる.
- 目的
 - (1)科学又は技術の発達のための実験
 - (2)電波の利用の効率性に関する試験
 - (3)電波の利用の需要に関する調査
 - これまで数ヶ月必要だったのに対して、数週間で取得を可能。
 - cf. 静大情報通信 準備一免許申請 1ヶ月 免許申請一免許交付 2週間

- 特定実験試験局の条件は、主に混信の防止を図る観点から次のとおり定められている。
 - (1)周波数、空中線電力及び使用可能な地域は、予め告示された範囲内とします。
 - (2)免許期間は、特定実験試験局が使用可能な周波数等を定める告示に規定する期間を超えない範囲で、最長5年です。
 - (3)登録点検事業者による無線設備の事前点検が必要です。
 - (4)混信を回避するため、特定実験試験局同士の運用調整が必要です。
- その他の簡略化：
 - 時計、無線検査簿及び無線業務日誌の備え付けの省略
 - 許可を要しない工事設計の軽微な事項の見直し、無線設備の設置場所の変更による検査の省略等の事後手続

申請に必要な書類と手続き

- 1. 書類
 - 1. 特定実験局開局申請
 - 収入印紙貼付
 - 2. 工事設計書
 - 登録点検事業者・東海総合通信局と打ち合わせながら作成
 - 3. 無線局事項書
 - 登録点検事業者・東海総合通信局と打ち合わせながら作成
 - 4. 登録点検結果通知書
 - 登録点検事業者による検査が必要
 - 無線従事者免許の確認, 記載
 - 検査自体は送信用ドータボード5台で丸一日
 - 5. 実験計画書
 - 目的, 具体的方法, 場所, 期間, 局数の根拠
 - 6. 情報通信研究グループ説明
 - 7. 無線従事者選任届(免許許可後)
- 2. 費用(1局当たり6700円)

登録点検

- 登録点検業者が行う作業
 - 無線機(ドーターボード)を無変調・変調のそれぞれで占有周波数帯幅, 出力周波数, 空中線電力, 帯域外領域, スプリアス領域などをスペアナで測定
- 登録点検で使ったプログラム
 - 無変調波
 - 簡単な送信用サンプルプログラム「tx_test.py」を利用
 - 変調波
 - benchmark_tx.pyを使用し, DBPSKでデータ送信
- 5台のXCVR2450を検査した. うち1台はスプリアスが大きく不適となり, 4台の申請となった 歩留80%
- その他のノウハウ
 - アンテナをつける位置, 無変調波の安定化, 他組織との調整
 - <http://www.mlab.t.u-tokyo.ac.jp/~saru/usrp/> 坂本覚え書き

静岡大学情報通信研究グループ

- 静岡大学
 - 渡辺 尚(代表)、水野 忠則、桑原 義彦、杉浦 彰彦、大内 浩司、石原 進、和田 忠浩、萬代 雅希、峰野 博史、石橋 功至、木谷 友哉、羽多野 裕之、山川 俊貴、椋本 介士、高柳 正勝
- ATR
 - 小花 貞夫、四方 博之
- 三菱電機
 - 渡辺 正浩
- 通菱テクニカシステム
 - 木崎 一廣
- 東京大学
 - 森川 博之、猿渡 俊介

総務省東海総合通信局実験局免許公示

<http://www.soumu.go.jp/soutsu/tokai/musen/tokutei/menkyo.html>

無線局免許状			
免許人の氏名又は名称	国立大学法人静岡大学 情報通信研究グループ 代表者 渡辺 尚		
免許人の住所	静岡県浜松市中区城北3-5-1		
無線局の種類	特定実験試験局	免許の番号	海実第2285号 海実第2288号
免許の年月日	平 21. 6. 10	免許の有効期間	平 25. 6. 30 まで
無線局の目的	実験試験用		運用許容時間 常 時
通信事項	研究に関する事項		
通信の相手方	免許人所属の特定実験試験局		
識別信号	しずだいじょうほうつうしんとくていじっけん 1~4		
無線設備の設置場所又は移動範囲	別紙のとおり		
電波の型式、周波数及び空中線電力	別紙のとおり		
備考	<small>(注) この周波数の使用は、他の無線局の運用に妨害を与えない場合に限る。 法律に別段の定めがある場合を除くほか、この無線局の無線設備を使用し、特定の相手方に対して行われる無線通信を傍受してその存在若しくは内容を漏らし、又はこれを窃用してはならない。</small>		
平成 21 年 6 月 10 日			
総 務 大 臣			